

Fundamentals of Digital Systems

by
Robert M. Laurie

This text material is a work in progress.
Please contact me with suggestions and corrections:

Attn: Robert M. Laurie
UMUC – Asia
Unit 5060 Box 0100
APO, AP 96328-0100

rlaurie@ad.umuc.edu

Copyright © 2001 by R.M. Laurie, All Rights Reserved.

Chapter 1. Introduction

Digital computers have brought about the information age that we live in today. Computers are important tools for humankind in that they can locate and process enormous amounts of information very quickly and efficiently. They allow us to utilize our mathematical disciplines to the fullest. In one second a computer can perform calculations that would take a person months to do by hand. However, computers are not creative and do only what we tell them. The list of instructions that tells the computer what to do is called a computer program.

System reliability, fast performance, and efficient information storage and retrieval are major factors in the acceptance and use of digital computer systems. The high reliability of computer systems is due largely to the fact that all data is in a digital format. Digital computers are designed such that digital formatted data can be processed quickly and efficiently.

1.1. Digital Logic States

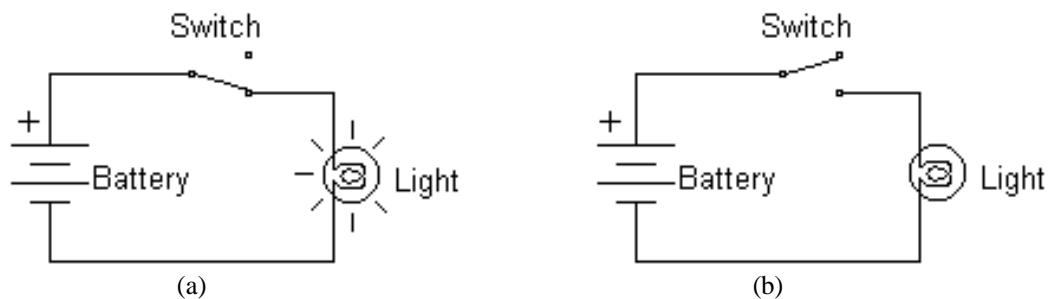
A computer is made up of many digital circuit modules that pass information in the form of *digital signals*. These signals can represent either program instructions or data to be processed. A digital signal can be considered a *logic variable* in that it can have only one of two possible values at any moment in time. These values are called *logic states*. Figure 1.1 describes the common notation for logic states.

Figure 1.1 Common Notation for Logic States

True	On	Closed	Yes	1	High	2 to 5 Volts
False	Off	Open	No	0	Low	0 to 1 Volt

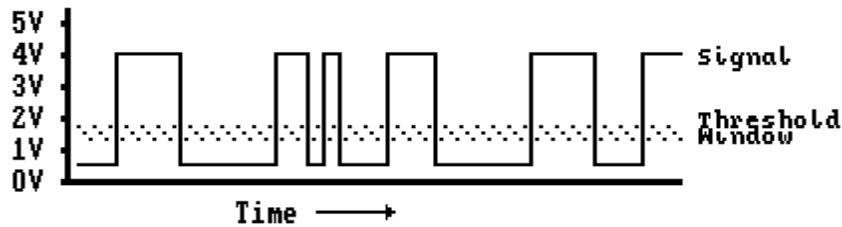
Two possible logic states can be represented by the opening and closing of the switch in the circuit of Figure 1.2. When a logic state is TRUE, the switch is CLOSED and the light goes ON. When a logic state is FALSE, the switch is OPENED and the light goes OFF. A question that can be answered with YES or NO can also be said to have two logic states.

Figure 1.2 Simple Digital Circuit Using a Switch



Digital circuit inputs and outputs are logic variables whose states are usually represented in binary (base 2) notation as a '1' or '0'. A digital signal is characterized as a time variant signal, which is in either a HIGH or LOW State. Transition time between HIGH and LOW states is minimized which results in a waveform as depicted in Figure 1.3. The primary constraint for a digital signal is whether the signal is above or below a specified threshold window. This threshold window is commonly between 1 and 2 Volts. Any signal which is greater than 2 Volts is considered logic '1' or HIGH state, and any signal which is less than 1 Volt is considered logic '0' or LOW state. Within the threshold window (between 1 and 2 Volts) the Logic State of the signal is undefined.

Figure 1.3 Digital Signal Representation



To perform logic operations, a device is required which can function as a switch with two possible states. Generally, it is desirable to have this switching occur as fast as possible. Electronics is currently the fastest, most compact, and least expensive way to implement a digital switch. Therefore, computer design is usually considered in the realm of electrical engineering.

Digital switches are implemented using an electronic device called a transistor. Using integrated circuit chip technology, it is possible to integrate millions of transistors on a single silicon chip. The rapid advances in digital electronics technology have led to the relatively inexpensive and compact computer systems that we use today.

The binary (Base 2) number system is often used to represent the states of several related logic variables at a specific instant in time. Figure 1.4 describes the binary equivalent for the decimal (Base 10) numbers zero through fifteen. Only two possible values can exist for each binary digit, either '0' or '1'. A single binary digit is called a *bit* and a group of eight bits is called a *byte*. A group of four bits as shown in Figure 1.4, is called a *nibble*.

Figure 1.4 Binary to Decimal Conversion Table (0 through 15)

Binary	Decimal	Binary	Decimal
0 0 0 0	00	1 0 0 0	08
0 0 0 1	01	1 0 0 1	09
0 0 1 0	02	1 0 1 0	10
0 0 1 1	03	1 0 1 1	11
0 1 0 0	04	1 1 0 0	12
0 1 0 1	05	1 1 0 1	13
0 1 1 0	06	1 1 1 0	14
0 1 1 1	07	1 1 1 1	15

Note that the binary number is zero-filled to four bits, just like the decimal number is zero-filled to two digits. A thorough description of the binary number system, including conversion methods, is presented in Chapter 5.

1.2. Modularity

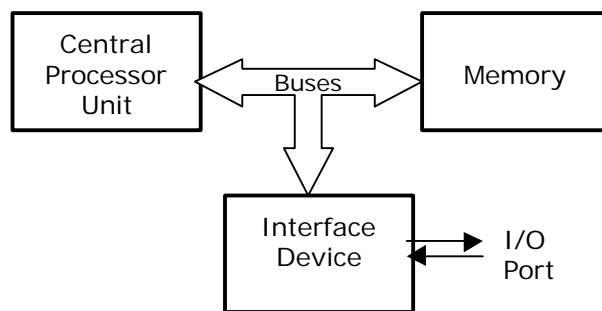
Two major aspects in the design of any computer system are hardware and software. *Hardware* is the physical computer, while *software* consists of the computer programs required to make the computer perform desired functions. Hardware and software must function together in a working computer system.

Modularity is one of the most important concepts in computer engineering and is applied in both hardware and software. Modularity is the process by which once something is designed to perform a desired function; it may be used as a functional block or module to implement the same function in a future design. Modularity avoids the "re-inventing of the wheel" syndrome, and minimizes design time. Modularity can best be utilized by partitioning a large task into simpler functional blocks or modules. This partitions the design into many manageable tasks that can usually be solved separately using previously designed modules. These modules are then interconnected to perform the desired function.

Hardware modularity has led to the rapid advances in computer technology. A computer can be viewed as a group of functional modules, with each module consisting of a hierarchy of smaller sub-modules. All modules are constructed using transistors in the form of integrated circuits. Figure 1.5 illustrates a block diagram of a computer system. All digital computer systems, whether small microprocessor based computers or large super computers, have this same structure. The major components are memory, a CPU, and I/O ports. Memory is used to store binary data that can represent either information or program instructions. The Central Processing Unit (CPU) is used to process data as specified by the instructions of the program. Input/output (I/O) ports are used to transfer information between interface devices and peripherals that may be in the form of a keyboard, display, printer, or disk drives. Figure 1.5 illustrates all functional blocks interconnected using circuit paths called buses.

Software modularity is utilized when a computer program is being written. A program can be divided into many functional modules or *functions*. Each function can be written and debugged separately. Thus, software development employs modularity principles by partitioning a large program into several smaller and more manageable tasks.

Figure 1.5 Computer System Block Diagram



Chapter 2. Combinational Logic

The hardware of a computer system is made of digital logic devices that are interconnected to form a digital circuit. Digital logic is divided into two categories: combinational logic and sequential logic. Combinational logic devices respond with a fixed set of transformation rules, which specify the state of all outputs for every combination of input states. For combinational logic the outputs are a function of only the present state of the inputs. For sequential logic the output is a function of both the present state of inputs and the past state of the output.

2.1. Logic Gates

A *logic gate* is a device that uses a fixed set of rules to transform a set of logical variable inputs into a single logical variable output. Although the inputs and output may vary with time, the transformation rules for a logic gate are time invariant. The output is dependent only on the states of the inputs at that instant.

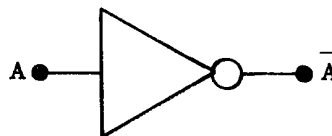
Logic gates can be connected together to form complex digital circuits, called *combinational networks*. These networks can transform a large set of digital inputs into a large set of outputs. When studying combinational networks, past behavior is not important; what is important are the rules that specify the state of the outputs as a function of all combinations of input states.

All digital circuits, including the largest of computers, are built from three primary logic gates. These primary gates are called *NOT* (or Inverter), *AND*, and *OR*. The inputs and output of a logic gate are logic variables in either a '1' or '0' state. The transformation rules for a gate are specified in a truth table. The *truth table* specifies the state of the output for all possible combinations of input states. Figure 2.1 contains the truth tables for the NOT, AND, and OR gates. Also shown is the digital circuit drawing symbols and Boolean algebra symbols for each of these three gates.

Figure 2.1 Truth Tables for Primary Gates. (a) NOT, (b) AND, and (c) OR

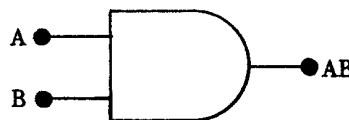
(a) **NOT (Inverter)**

A	$\bar{A} = \text{NOT } A$
0	1
1	0



(b) **AND**

A	B	$AB = A \text{ AND } B$
0	0	0
0	1	0
1	0	0
1	1	1



(c) **OR**

A	B	$A+B = A \text{ OR } B$
0	0	0
0	1	1
1	0	1
1	1	1



The NOT gate is often called an *inverter* since its output state will be the inverse of the input state. The circuit symbol for the NOT gate is a triangle with a circle on the output, as shown in Figure 2.1a. The Boolean algebra symbol for the NOT operation is an inversion bar placed over the input logic variable. The NOT operation is defined for only one input, while the AND and OR operations require more than one input.

The AND operation for two inputs is depicted by the truth table and logic symbol shown in Figure 2.1b. The AND gate output is '1' only if all inputs are '1'; otherwise, the output is '0'. The Boolean algebra symbol for the AND operation is the same as multiplication in algebra of real numbers.

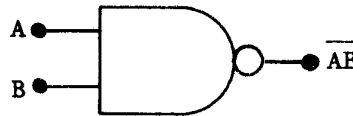
The OR operation for two inputs is depicted by the truth table and logic symbol illustrated in Figure 2.1c. The OR gate output will be '1' if any input is in the '1' state. Therefore, the only case for which the output is '0' occurs when all inputs are '0'. The Boolean algebra symbol for the OR operation is represented by a plus symbol.

Three additional gates, the *NAND*, *NOR*, and *XOR* (Exclusive OR) are simple combinations of the three primary gates NOT, AND, and OR. Figure 2.2 contains the truth tables for the NAND, NOR, and XOR. Also shown, are the digital circuit symbols and Boolean algebra symbols for each of these three gates.

Figure 2.2 Truth Tables for (a) NAND, (b) NOR, and (c) XOR Gates.

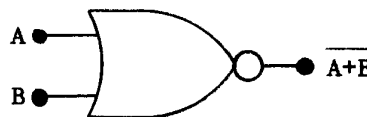
(a) **NAND**

A	B	$\overline{AB} = A \text{ Not AND } B$
0	0	1
0	1	1
1	0	1
1	1	0



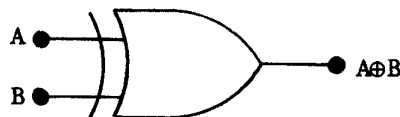
(b) **NOR**

A	B	$\overline{A+B} = A \text{ Not OR } B$
0	0	1
0	1	0
1	0	0
1	1	0



(c) **XOR (Exclusive OR)**

A	B	$A \oplus B = A \text{ XOR } B$
0	0	0
0	1	1
1	0	1
1	1	0



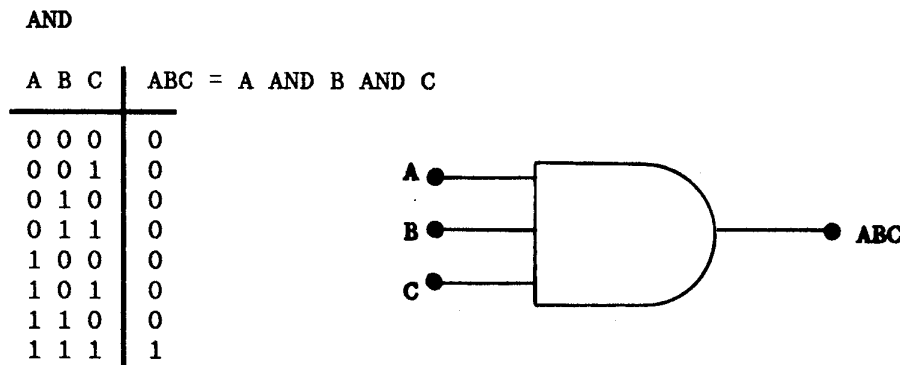
The NAND gate can be considered an abbreviation for NOT-AND. The truth table for a NAND gate is the same as an AND gate with its output inverted (NOT). The NAND gate can be drawn as a digital circuit with an AND gate output connected to a NOT gate input. The NAND gate is such a common logic gate that it is usually drawn as an AND gate with a circle on the output of the gate as shown in Figure 2.2a. This circle is called an inversion circle and represents a NOT operation performed on the specified output. The Boolean algebra symbol for a NAND operation is represented with the inversion of the AND operation or \overline{AB} .

The NOR gate is similarly a NOT-OR operation which is represented as an OR gate with its output inverted. This can be verified by examining the truth table of Figure 2.2b. The circuit symbol for a NOR gate is an OR gate with an inversion circle on the output. The NOR operation is represented in Boolean algebra as $\overline{A+B}$.

The Exclusive OR (XOR) operation is defined by the following statement: If an odd number of inputs are '1' then the output is a '1'; otherwise, the output is '0'. The circuit symbol for an XOR gate is shown in Figure 2.2c. It is similar to an OR gate with an additional arc drawn across the input side of the gate. The Boolean algebra symbol for the Exclusive OR operation is a plus sign enclosed in a circle denoted by $A\oplus B$.

More than two inputs may be used for the AND, NAND, OR, NOR, and XOR gates. A gate with a total of n inputs will have 2^n possible combinations of these n inputs. Therefore, when constructing the truth table for a gate with n inputs, 2^n rows must exist which represent all possible combinations of input states. This is illustrated by the truth table in Figure 2.3 for a 3-input AND gate. It is best to use the binary counting scheme described in Figure 1.4 to account for all possible combinations of input states.

Figure 2.3 Truth Table for a 3-Input AND Gate



2.2. Boolean Algebra

Boolean algebra is a branch of mathematics for which the values of all variables are either '0' or '1'. There are three primary Boolean algebra operations which consist of the logic operations NOT, AND, and OR. Since binary (base 2) numbers can represent all logic values in digital circuits, Boolean algebra can be applied for analysis and synthesis of digital circuits. Boolean symbols were used for the output variable in the previously described truth tables. These symbols are summarized in Figure 2.4.

Figure 2.4 Boolean Symbols

\overline{A} = NOT A = Complement A	$A\oplus B$ = A Exclusive-OR B
AB = A AND B	$\overline{A B}$ = A NAND B
$A+B$ = A OR B	$\overline{A+B}$ = A NOR B

Just as any other branch of mathematics, Boolean algebra has many identities that have been proven and can be used for simplification or to find equivalent expressions. The most common identities are listed in Figure 2.5. All variables in Boolean algebraic descriptions are logic variables. Therefore, the variables A, B, and C of Figure 2.5 have a value of either '1' or '0'. Keeping this in mind, many of the results of these identities are fairly intuitive. The first five identities of Figure 2.5 are the fundamental

identities of Boolean algebra. Using these identities, all other identities of Figure 2.5 can be proven using Boolean algebraic manipulation.

Once a Boolean algebra equivalency is proven it can be used as an identity. Equivalence in Boolean algebra is not the same as equality in algebra of real numbers. For example, consider an OR gate with both inputs connected to logic '1'. This could be written as $1 \text{ OR } 1 = 1$ or in Boolean algebraic form as $1 + 1 = 1$. A common mistake during Boolean algebra manipulation is the improper use of the inversion bar. Note that $\overline{A B} \neq \overline{A} \overline{B}$.

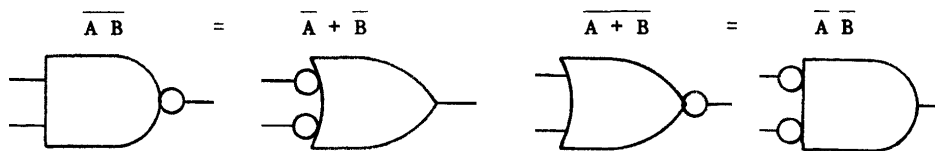
DeMorgan's Law is a very important identity that is used for manipulating inversion bars. DeMorgan's AND Law is written in Boolean algebraic form as $\overline{A B} = \overline{A} + \overline{B}$, and can be stated as NOT the quantity A AND B is equivalent to NOT A OR NOT B. DeMorgan's OR Law is an alternate form and is written algebraically as $\overline{A+B} = \overline{A} \overline{B}$. Both the AND and OR forms of DeMorgan's Law are equivalent, which is proven in Example 2.2d. DeMorgan's Law can also be extended to more than two variables. By applying DeMorgan's law, equivalent gate symbols can be found as shown in Figure 2.6. Note that inversion circles can be used to show an inversion of the input variables as well as the output.

Duality is a Boolean algebra principle describing once equivalence is proven a dual of this equivalence can be determined that is also a valid equivalence expression. The *dual* of a Boolean expression can be determined as follows: Replace all AND operations with OR operations and all OR operations with AND operations on both sides of the equivalency; then replace all 1's with 0's and 0's with 1's on both sides of the equivalency. The duality principle was applied in constructing the identity table of Figure 2.5. Note that the dual of the AND form of an identity is the OR form of the same identity.

Figure 2.5 Boolean Algebra Identities

Name	AND Form	OR Form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \overline{B}$
Absorption law	$A(A + B) = A$	$A + AB = A$
Inclusion law	$A(\overline{A} + B) = AB$	$\overline{AB} + B = A + B$

Figure 2.6 Gate Description for DeMorgan's Laws



Substitution is the process by which a logic variable may be substituted for a Boolean expression or vice-versa. Substitution is frequently used when simplifying a Boolean expression. It can also be used to extend the identities of Figure 2.5 to a greater number of variables than what is specified.

Double Inversion is another identity. If a Boolean expression has two inversion bars over it, they cancel each other and both inversion bars can be removed. This relationship can be expressed in algebraic form as $\overline{\overline{A}} = A$ or $\overline{\overline{A B}} = A B$. Note that $\overline{A B} \neq \overline{A} \overline{B}$.

2.3. Boolean Equivalence Verification

For any given combinational network, a Boolean expression can be written for each of the outputs in terms of the inputs. Using Boolean algebra, equivalent circuit designs can be found for optimizing a design or predicting the results for various situations.

Two methods may be employed to verify Boolean identities or find an alternate equivalent solution. One method uses a truth table and the other uses Boolean algebra.

2.3.1. Truth Table Verification

When constructing a truth table the output values must be found for all possible combinations of input states. For n input variables, 2^n rows will be required for the truth table. All logic variables are in either the '1' or '0' state. Example 2.1a through Example 2.1c illustrates the use of truth tables to prove equivalency of two Boolean expressions.

Example 2.1 Truth Table Verification

a) Verify: $1A = A$ (Identity AND Law)

A	1	1A
0	1	0
1	1	1

↑ Verifies

b) Verify: $A + B = \overline{\overline{A} \overline{B}}$ (De Morgan's OR Law)

A	B	A + B	$\overline{\overline{A} \overline{B}}$	\overline{A}	\overline{B}	$\overline{A} \overline{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

↑ Verifies

c) Verify: $(A + B)(A + \overline{B}) = A$

A	B	A + B	A + \overline{B}	$(A + B)(A + \overline{B})$
0	0	0	1	0
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

↑ Verifies

Construction of the truth table begins by writing all possible combinations of input logic states in the left-most columns of the truth table. This can be done best by using the binary counting scheme of Figure 1.4 to account for all possible combinations of input variables. Then perform one primary logic operation (AND, OR, or NOT) by determining the output value of the logic operation for all combinations of input logic variables. Any column can be used as an input to perform additional logic operations, whether it is an output from a previous logic operation or an input logic variable. For clarity, vertical lines should separate all columns that represent results of logic operations. Continue performing logic operations in the

truth table until two columns exist which represent the two Boolean expressions on either side of the equality. If the columns match for all possible combinations of the inputs, the two Boolean expressions are said to be equivalent.

Example 2.1 demonstrates truth table verification to prove the Identity AND Law and DeMorgan's OR Law. Any of the identities of Figure 2.5 may be proven using truth tables. Example 2.1c describes a Boolean relationship, which is verified using a truth table. Once this relationship is proven to be equivalent, it may be used as an identity.

2.3.2. Boolean Algebra Verification

In mathematics, algebraic manipulation using proven identities can create equivalent expressions. The same procedure can be applied to Boolean algebra. Using the Boolean identities of Figure 2.5, substitution, and double inversion, a Boolean expression can be manipulated to find an alternate equivalent expression. Two expressions are equivalent when they generate the same results for all possible combinations of logic variable inputs. Equivalence can be proven using truth tables. However, for expressions with more than four logic variables, truth tables become tedious and Boolean algebra verification is often easier.

Example 2.2a and Example 2.2b demonstrate the use of Boolean algebra to verify the Absorption OR Law and Inclusion OR Law using other identities from Figure 2.5. Note that the identities of Figure 2.5 specify the form of equivalent expressions and not the actual input variables themselves. Example 2.2c illustrates the use of Boolean algebra to simplify an expression. After simplification is performed, one can see that output X is dependent only on the value of input C and is in fact equivalent to C. Example 2.2d demonstrates that both the AND and OR forms of DeMorgan's Law are equivalent using Boolean algebraic manipulation.

Example 2.2 Algebraic Verification

a) Verify: $A + A B = A$ (Absorption OR Law)

$$= A 1 + AB \quad (\text{Identity AND Law})$$

$$= A (1 + B) \quad (\text{Distributive OR Law})$$

$$= A 1 \quad (\text{Null OR Law})$$

$$= A \quad (\text{Identity AND Law})$$

b) Verify: $A\bar{B}+B = A+B$ (Inclusion OR Law)

$$= B+A\bar{B} \quad (\text{Commutative OR Law})$$

$$= (B + A)(B + \bar{B}) \quad (\text{Distributive AND Law})$$

$$= (B + A) 1 \quad (\text{Inverse OR Law})$$

$$= B + A \quad (\text{Identity AND Law})$$

$$= A + B \quad (\text{Commutative OR Law})$$

$$\begin{aligned}
\text{c) Simplify: } X &= \bar{A} \bar{B} C + A C + B C \\
&= C (\bar{A} \bar{B} + B + A) && \text{(Distributive OR Law)} \\
&= C (\bar{A} + B + A) && \text{(Inclusion OR Law)} \\
&= C (1 + B) && \text{(Inverse OR Law)} \\
&= C (1) && \text{(Null OR Law)} \\
X &= C && \text{(Identity AND Law)}
\end{aligned}$$

d) Show the two forms of De Morgan's Law are equivalent.

$$\overline{A B} = (\bar{A} + \bar{B}) \quad \text{(De Morgan's AND Law)}$$

$$\overline{\bar{A} \bar{B}} = \overline{(\bar{A} + \bar{B})} \quad \text{(Invert Both Sides)}$$

$$A B = \overline{(\bar{A} + \bar{B})} \quad \text{(Double Inversion)}$$

$$\text{Set } C = \bar{A} \quad D = \bar{B} \quad \text{(Substitution)}$$

$$\bar{C} \bar{D} = \overline{C + D} \quad \text{(De Morgan's OR Law)}$$

2.4. Combinational Network Design

Design of combinational networks requires a working knowledge of Boolean algebra, truth table construction, and digital logic gate representations. It is essential to know how to go from one representation to another.

Several configurations of logic gates may have the same input/output characteristics; that is, each combination of input states produces the same output states. Two combinational networks with the same input/output characteristics are said to be equivalent. Equivalence is verified through truth table construction or Boolean algebra. Figure 2.7 illustrates two equivalent combinational networks. The networks are the digital circuits representing the Distributive AND Law of Figure 2.5.

2.4.1. Description to Digital Circuit Design

Once a description of the desired logic function is defined, a digital circuit can be designed to implement the function. The description may be a truth table or a verbal description. To understand the function of a digital circuit, the state of the outputs must be known for all possible combinations of inputs. Constructing the truth table, as discussed in Section 2.3.1, is one way to account for all possible combinations of inputs.

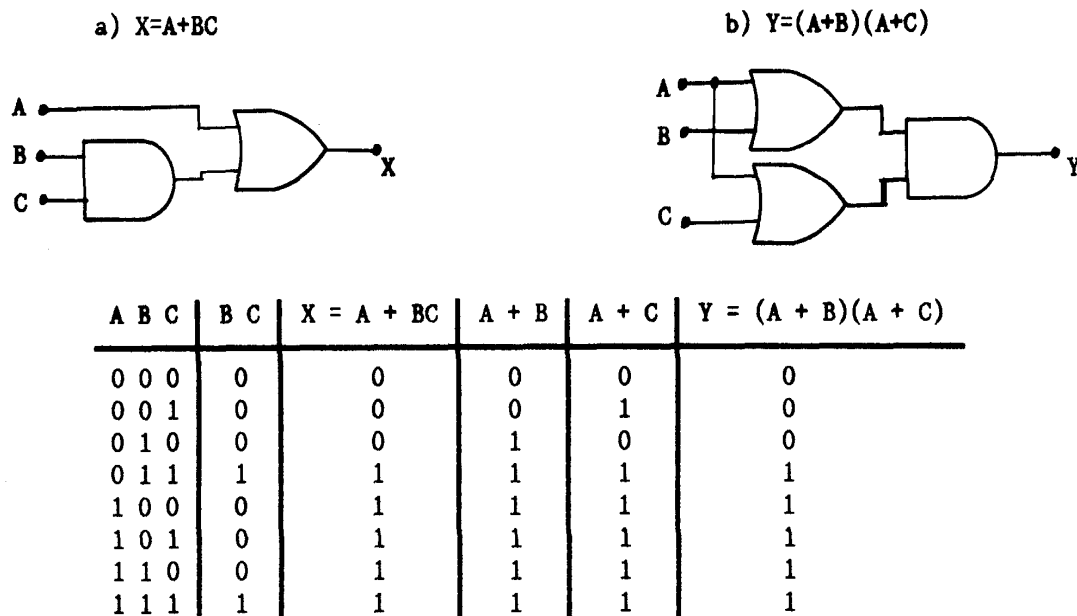
After the description is defined, the next step towards designing the digital circuit is to determine a Boolean expression that describes the desired function. The Sum of Products method can be utilized to determine a valid Boolean expression directly from a truth table description. The induction method is another method used to determine a valid Boolean expression directly from a verbal description.

The truth table of Figure 2.7 is used as an example to demonstrate the sum of products method for the second output column ($X=A+BC$).

After a Boolean expression has been found to describe the logic function, the digital circuit can be designed directly. This is accomplished by using the gates discussed in Section 2.1 to implement each of the logic functions of the Boolean expression. Just as with algebra of real numbers, product operations (AND functions) are performed first and then sum operations (OR functions). Parentheses are used to specify a different order of operation and to group terms.

Two simple digital circuits are illustrated in Figure 2.7 for outputs X and Y. Examining the output X and Y columns of the truth table of Figure 2.7, one can verify that these are equivalent circuits.

Figure 2.7 Equivalent Combinational Networks



2.4.1.1. Sum of Products (SOP) Method

The Sum of Products (SOP) method is a procedure, for determining a valid Boolean expression from a truth table description. This method considers only those rows of the truth table with logic '1' in the output column.

Examining the truth table of Figure 2.7, Output X equals '1' in the fourth row. This occurs when the inputs conditions are $A=0$, $B=1$, and $C=1$. Similarly, for the fifth row X equals '1' when $A=1$, $B=0$, and $C=0$. Boolean expressions can be written to express these relationships for the fourth and fifth rows as $\overline{A}BC=1$ and $A\overline{B}\overline{C}=1$. Likewise, Boolean expressions can be written for the sixth row as $A\overline{B}C=1$, the seventh row as $A\overline{B}\overline{C}=1$, and eighth row as $ABC=1$. These Boolean expressions are called *minterms*. Minterms only exist for rows that have an output of logic 1. A Boolean expression can be written describing the entire truth table by OR'ing each of the five minterms together. This will result in the following Boolean expression:

$$X = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + A\overline{B}\overline{C} + ABC$$

Notice the form of the expression is a sum of products; hence the name of the method describes the resultant form.

The digital circuit can then be created from this sum of products expression as illustrated in Figure 2.8.

2.4.1.2. Induction Method

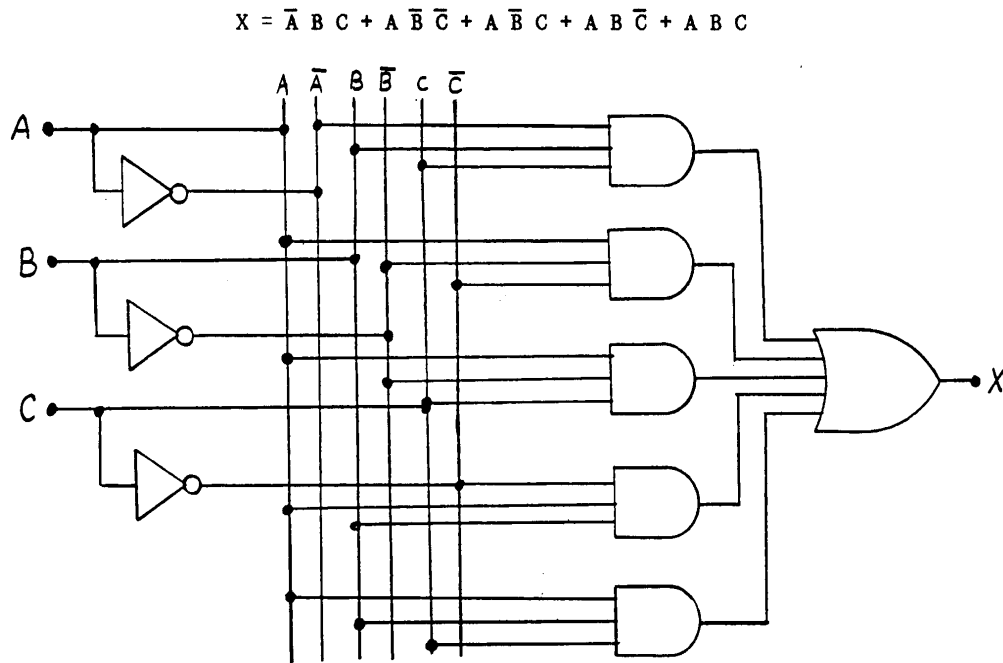
The induction method describes the process of determining a Boolean expression directly from the verbal description. This is particularly useful for systems with more than four inputs because truth tables become cumbersome and the resulting sum of products expressions can become lengthy. Output X in Figure 2.7 can be described verbally as:

X is 1 if A is 1, or if B and C are 1. Otherwise, X is 0.

From this verbal description the Boolean expression can be written directly:

$$X = A + BC$$

Figure 2.8 SOP Digital Circuit Drawing



2.4.2. Digital Circuit Minimization

The digital circuits of Figures 2.7 are 2.8 are all equivalent combinational networks since they all generate the same input/output relationships and are derived from the same truth table. The digital circuit of Figure 2.7a is the preferred choice since it requires the least number of logic gates for the implementation.

Once a Boolean expression is found using sum of products or induction, algebraic simplification of the expression is often possible using Boolean identities. This may greatly reduce the number of gates required to construct the circuit.

As an example, suppose a digital circuit must be designed which has two outputs X and Y; and must accomplish the following functions.

Output X is 1 if either E or F are 1 and D is 0. Otherwise X is 0.

Output Y is 1 if A and B and C are 1, or D is 0 and either B or C are 0, or if D is 1.

Otherwise Y is 0.

This circuit has six inputs (A, B, C, D, E, and F) and two outputs (X and Y). Constructing the truth table would be tedious with 64 rows, and the resulting SOP Boolean expressions would require much effort to reduce. For these reasons, the Boolean expression is found using induction from the verbal description and then simplifying the result.

Both the unsimplified and simplified digital circuits for this example are illustrated in Figure 2.9 and Figure 2.10. Note that inversion circles are shown on the inputs of some of the gates. These symbolize inverters.

$$X = (E + F) \bar{D}$$

$$Y = A B C + \bar{D} (\bar{B} + \bar{C}) + D$$

$$= A B C + (\bar{B} + \bar{C}) + D \quad \text{(Inclusion OR Law)}$$

$$= A B C + \overline{B C} + D \quad \text{(De Morgan's AND Law)}$$

$$= A + \overline{B C} + D \quad \text{(Substitution and Inclusion OR Law)}$$

Figure 2.9 Unsimplified Digital Circuit From Induction

$$X = (E + F) \bar{D} \quad Y = A B C + \bar{D} (\bar{B} + \bar{C}) + D$$

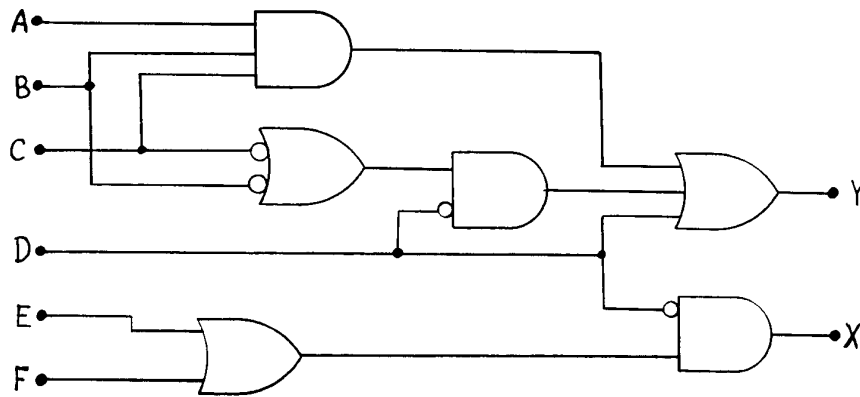
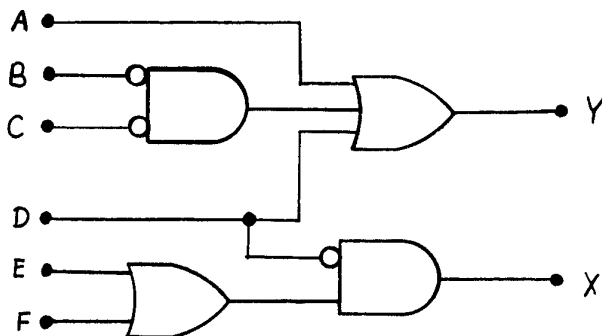


Figure 2.10 Simplified Digital Circuit from Induction

$$X = (E + F) \bar{D} \quad Y = A + \bar{B} C + D$$



2.4.3. Boolean Expressions from Digital Circuit

Modification of existing digital circuits requires the ability to go from a digital circuit drawing to a Boolean expression. Two procedures for accomplishing this conversion are discussed in this section. The first procedure is used when Boolean expressions for all outputs of a digital circuit are required. The second procedure is for the case when the Boolean expression for only one output is needed.

The procedure for determining the Boolean expression for all outputs in a digital circuit is quite straightforward. Beginning from the input side of the digital circuit drawing, write the output Boolean expression as you progress through each gate. Then use the output equation from the preceding gate as the input to the next gate in the circuit. Continue writing Boolean expressions at the output of each gate until the Boolean expressions for all outputs of the digital circuit are determined. An example of this procedure is shown in Figure 2.1. Notice the 2-input NAND gate with both inputs tied together. A NAND gate in this configuration will function as an inverter. Similarly, a NOR gate with all inputs tied together would also function as an inverter. This can be verified by examining the truth tables of Figure 2.2.

Consider the case of a complex digital circuit with many outputs, and suppose the Boolean expression for only one output is needed. For this case, the first step is to trace through the circuit from the output of interest to the inputs, marking each gate that will affect this output. This is illustrated in Figure 2.9. After the gates are marked, proceed from the input side to the output of interest by writing the Boolean expression for the output of each affecting gate. This procedure will save considerable time when determining the Boolean expression of a single output in complex combinational networks.

Once the Boolean expression is determined for an output, Boolean algebra can then be used to reduce the expression to a simplified equivalent form.

Figure 2.11 Boolean Expressions for All Outputs

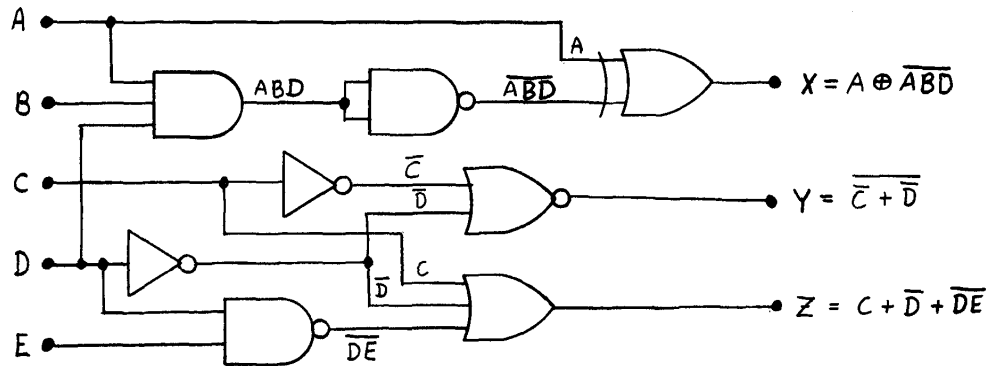
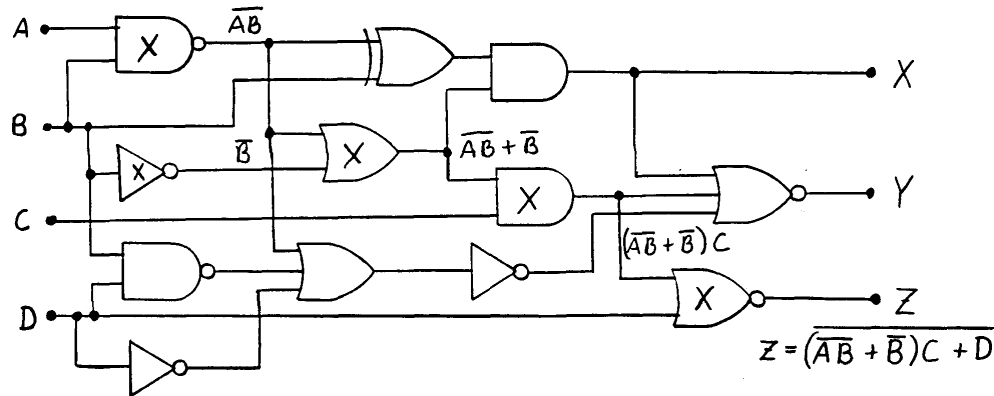


Figure 2.9 Boolean Expression For One Output



2.5. Common Combinational Circuits

Once a digital circuit has been designed using individual gates to perform a specific function, it is often desirable to use the newly created circuit as a module for future designs. Discussed in this section are several commonly used combinational circuits that include decoders, multiplexers, adders, and arithmetic logic units. These combinational circuits are used as building blocks to construct all computers.

2.5.1. Decoders

A *decoder* is a digital circuit with n inputs and 2^n outputs. Figure 2.10a illustrates a logic gate diagram that can be used to construct the 3 to 8 decoder. Figure 2.10b is a block diagram of the 3 to 8 decoder. Note that the block diagram contains all inputs and outputs illustrated in the logic gate diagram. After a digital circuit has been designed to perform a specific function, it can be considered as a functional module and is usually represented by the block diagram.

The decoder of Figure 2.10 functions such that one and only one output is in the 1 state, as selected by a binary code placed on select inputs A, B, and C. Since only one output can be in the 1 state, all other outputs will be in the 0 state.

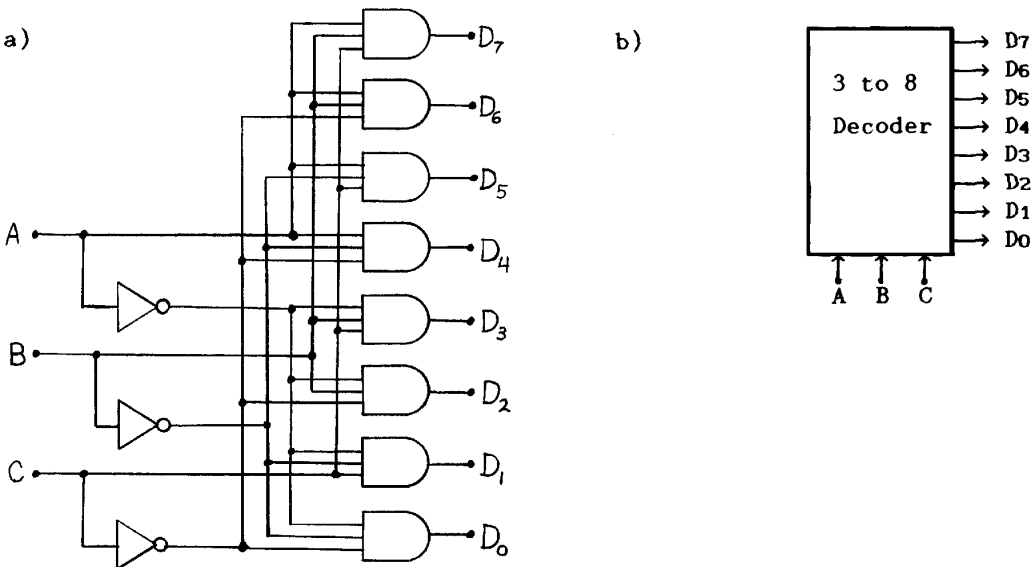
Decoders are used to select one of 2^n devices with an n -bit code. The n -bit code is often called the address of the selected device. Note that only one device can be addressed at a time since only one decoder output will be in the 1 state. Decoders are used in computers to select one of several functions in the CPU and to select one of many memory locations.

Decoders can be constructed to generate either positive logic outputs or negative logic outputs. The decoder of Figure 2.10 is an example of a positive logic output decoder. Positive logic output decoders

function such that the selected (or enabled) output is in the 1 state and all other outputs are in the 0 state (or disabled). Negative logic output decoders function such that the selected output is in the 0 state while all other outputs are in the 1 state. A negative logic output 3 to 8 decoder can be constructed by modifying the logic diagram of Figure 2.10. If the eight three-input AND gates are replaced with eight three-input NAND gates, then the decoder will have negative logic outputs.

Figure 2.10 3 to 8 Decoder

a) Digital Circuit b) Block Diagram



2.5.2. Multiplexers

A *multiplexer* is a combinational circuit in which one of several data inputs is selected and routed to a single output. Figure 2.11 is an example of a four data input multiplexer. The block diagram of Figure 2.11a illustrates the four data inputs (D_0 through D_3) of which one is selected and routed to the single output F . The data input is selected by applying a binary code on the select inputs A and B . Two select inputs will generate four unique binary codes. Therefore, any one of four data inputs may be selected when using two select inputs. For a general case multiplexer with n select inputs a maximum of 2^n data inputs could be available. A multiplexer with 4 select inputs may have a maximum of 16 data inputs.

The 4-data input multiplexer can be constructed using a 2 to 4 decoder and logic gates as shown in Figure 2.11b. The decoder can be considered as a sub-module within the multiplexer module. A logic gate implementation for the 4-data input multiplexer is illustrated in Figure 2.11c. The reader is urged to verify the multiplexer operation. A functional truth table for the 4-data input multiplexer is shown in Figure 2.11d. Note that the output state is the state of the selected data input.

Multiplexers are generally used for data routing applications and can be considered as a data switch. As an example, consider the case of four computers that are connected to one printer. Only one computer can send data to the printer at a time; therefore, a 4-data input multiplexer is chosen as a data switch. A two bit binary code will be used to address each of the four computers.

A demultiplexer performs the opposite function. One data input is routed to several possible data outputs. Select inputs determine which data output transmits the data. A demultiplexer will have one data input, n select inputs, and 2^n data outputs.

2.5.3. Binary Adders

Addition of binary numbers is accomplished using a digital circuit called an adder. Figure 2.12 illustrates a half adder that is used to add two single-bit binary numbers represented by logic variables A and B . The half adder circuit has two inputs A and B , and two outputs Sum and Carry.

Figure 2.11 4-Data Input Multiplexer

a) Block Diagram b) Digital Circuit c) Logic Gate Diagram d) Truth Table

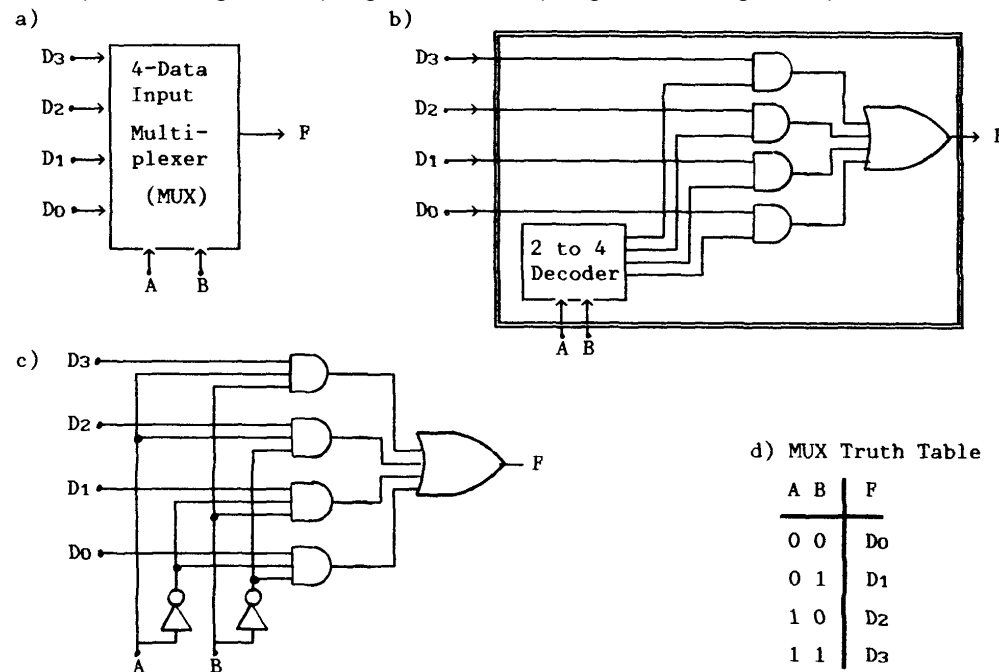
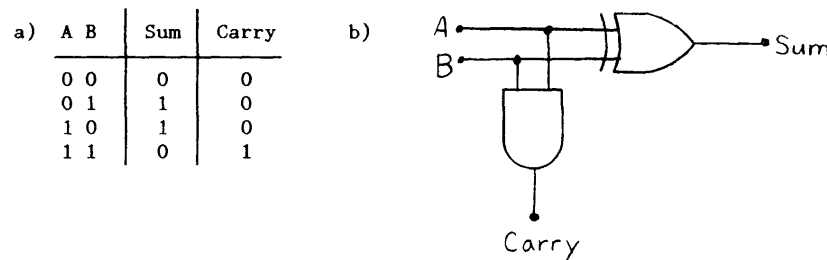


Figure 2.12 Half Adder

a) Truth Table b) Logic Diagram



The truth table for the half adder, shown in Figure 2.12a, describes binary addition of two single-bit binary numbers. When both inputs are 0, the sum is 0. If either input A or B is 1, but not both, the sum is 1. When both A and B are 1, the sum exceeds what can be shown with a single bit; therefore, the sum is 0 and the carry is set to 1. The carry output is 1 only when both inputs A and B are logical 1. Based on this description, the sum operation can be accomplished by using an Exclusive-OR gate and the carry operation can be performed using an AND gate.

To utilize the carry for the next significant bit, a full adder is used for multi-bit addition. The truth table and logic diagram of the full adder is shown in Figure 2.13. The 3 inputs A, B, and Carry In are added together to generate the two outputs Sum and Carry Out. Figure 2.14 illustrates a digital circuit, which will perform binary addition on two four-bit binary numbers, which are represented by A₃ to A₀ and B₃ to B₀. A₃ is the most significant bit and A₀ is the least significant bit of the 4-bit binary number A. Four full adder circuits (shown as block diagrams) are utilized to construct this digital circuit.

The Carry In of the least significant bit is connected to ground because a carry will not occur into the least significant bit. The Carry Out of the least significant bit is connected to the Carry In of the next significant bit as illustrated in Figure 2.14. The connections of Carry Out to Carry In of the next significant bit continue throughout the circuit. When the Carry Out of the most significant bit is 1, the sum exceeds what can be shown with the number of bits allotted for the sum. The Carry Out of the most significant bit is often called the Carry Flag.

Figure 2.13 Full Adder

a) Truth Table b) Logic Diagram

a)	A	B	Carry In	Sum	Carry Out
	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	0
	0	1	1	0	1
	1	0	0	1	0
	1	0	1	0	1
	1	1	0	0	1
	1	1	1	1	1

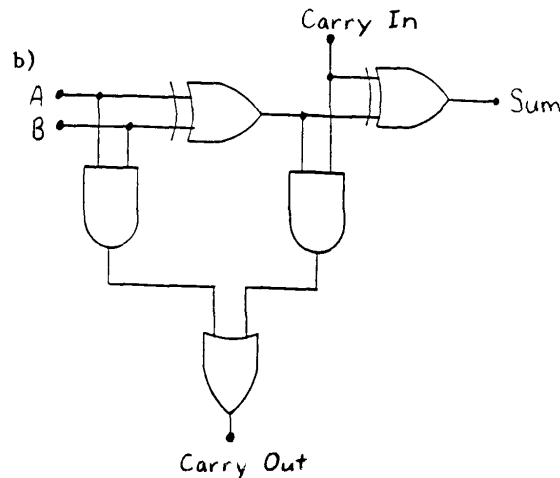
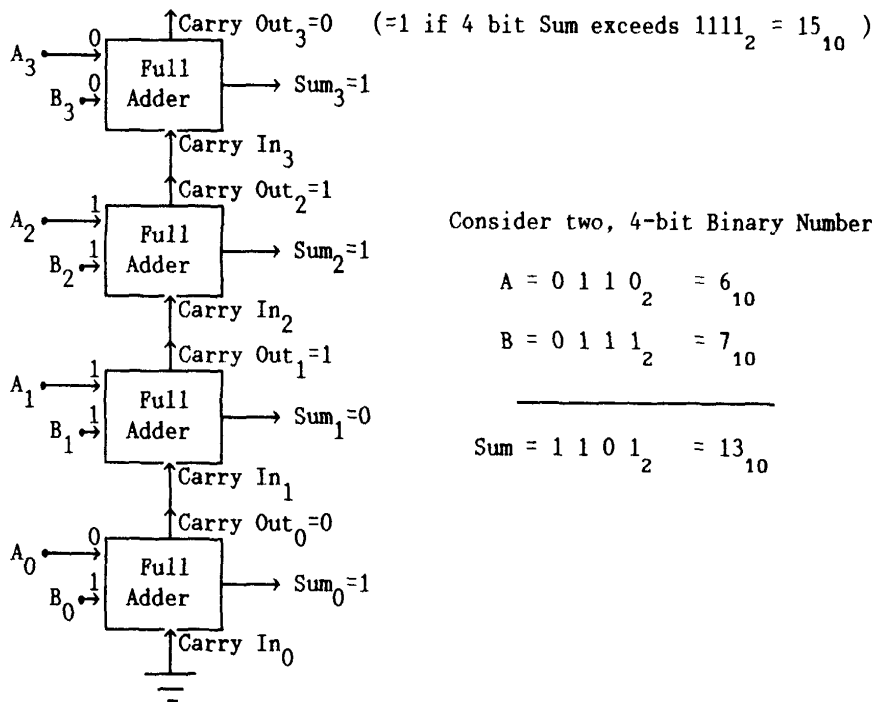


Figure 2.14 Four Bit Binary Adder



2.5.4. Arithmetic Logic Units

The Arithmetic Logic Unit (ALU) is a device, which can perform several operations on two binary numbers. All computers contain an ALU, as a module within the Central Processing Unit. The ALU performs the arithmetic and logic operations specified by the instructions of a computer program.

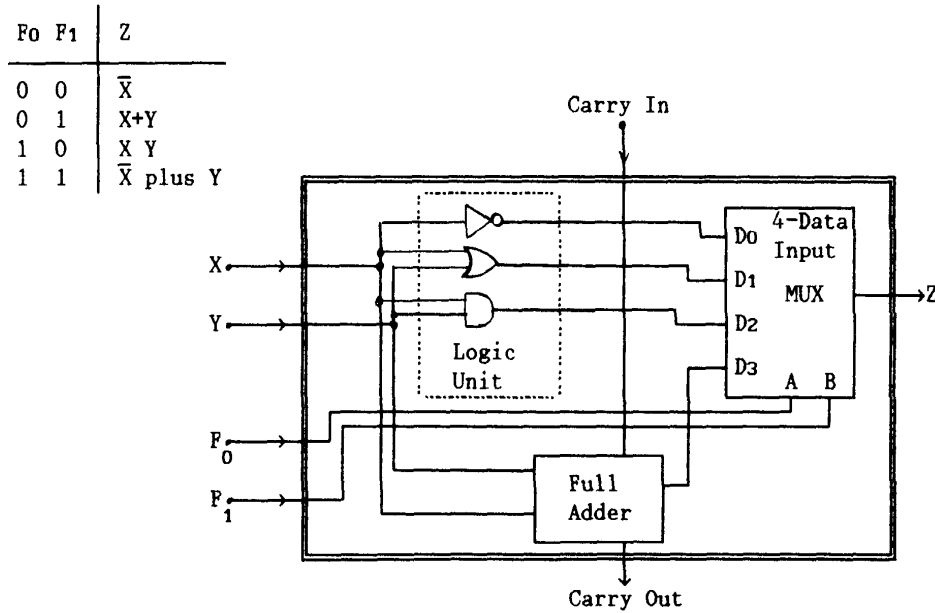
Figure 2.15 is an example of a single-bit 4-function ALU. Inputs X and Y are the two Boolean variables on which a particular operation is performed. Output Z is the result of the operation. The ALU shown will perform one of four functions (NOT, OR, AND, or SUM) on the inputs X and Y to generate the result Z. The function performed is determined by the values of functional inputs F₀ and F₁. For example, when the F₀ and F₁ inputs are 0 and 1 respectively, an OR operation will be performed on inputs

X and Y. When both F_0 and F_1 are 1, a sum operation is performed and the Carry In input and Carry Out output will be utilized. The functional truth table of Figure 2.15 summarizes these operations.

The ALU of Figure 2.15 can be thought of as a module, which contains three sub-modules, namely a full adder, a logic unit, and 4- data input multiplexer. The full adder and logic units are used to perform sum and logic operations on the input variables. The multiplexer is used to route the output of the selected function to output Z.

A four-bit ALU could be constructed by cascading four single-bit ALU modules together much like the full adder circuit of Figure 2.14. All four functional inputs F_1 would be connected together and all F_0 inputs would be interconnected so that each ALU would perform the same function on each bit.

Figure 2.15 Single-Bit 4-Function ALU



Problem Set

1. Make a conversion table from Decimal (Base 10) to Binary (Base 2) from zero to 35. Zero fill the binary numbers to generate six bits.
2. Draw the following gates and construct the truth tables for these gates.
 - a) 2-input AND Gate
 - b) 2-input XOR Gate
 - c) 2-input NOR Gate
 - d) 3-input NAND Gate
 - e) 3-input XOR Gate
 - f) 4-input OR Gate
3. Draw the gates representing the following Boolean symbols.
 - a) $A+B$
 - b) ABC
 - c) $\overline{A+B+C}$
 - d) $A\oplus B$
 - e) \overline{ABCD}
4. Prove the following Boolean Identities using Truth Tables.
 - a) $0A = 0$
 - b) $1 + A = 1$
 - c) $AA = A$
 - d) $A(A + B) = A$
 - e) $(A + B) + C = A + (B + C)$
 - f) $A(\overline{A} + B) = AB$
 - g) $A + BC = (A + B)(A + C)$
 - h) $\overline{AB} = \overline{A} + \overline{B}$
5. Prove the following equivalencies using Boolean algebra.
 - a) $(A + B)(A + C) = A + BC$
 - b) $A(\overline{A} + B) = AB$
 - c) $(A + B + C) \overline{A} = \overline{A+B+C}$
 - d) $\overline{A B + C + \overline{A} B C D + C C} = A B + C$
 - e) $A B (B+C) = \overline{A} + B$
 - f) $A \overline{B} + A \overline{C} + B C = A + B C$
 - g) $\overline{\overline{A B A C}} = A + B + C$
 - h) $\overline{(\overline{A} + \overline{B} + \overline{C})} C = A B + \overline{C}$

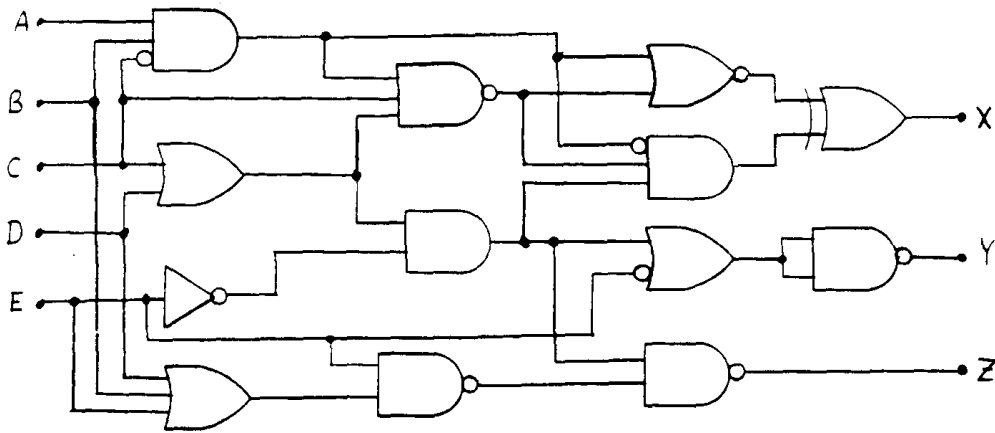
6. Write both sum of products Boolean expressions for output Z using the truth table shown. Draw a logic diagram for the Boolean expression.

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

7. Draw a logic diagram for the following Boolean Expression.

$$Z = \overline{(A \oplus B) + \overline{B}C + \overline{B}C\overline{C}}$$

8. Write a Boolean Expression for Output Z.



9. Construct the truth table for the 3 to 8 decoder of Figure 2.14a.

10. Construct a 4-bit adder using logic gates.

Chapter 3. Integrated Circuits

Digital circuit design is considered a high-level design method, because only the states of the inputs and outputs are important. Gates will function as expected as long as nominal analog circuit parameters are not exceeded. There are many advantages to using digital circuits over conventional analog circuits. These advantages include modularity, reliability, and noise immunity.

Several procedures have been discussed for designing a digital circuit from either verbal or truth table descriptions in Section 2.4.1. The next step is to use actual digital circuit components to construct the circuit.

Gates are not manufactured individually but are sold in packages containing several gates. These packages are called integrated circuits. Often the term integrated circuit is abbreviated IC or called by its nicknamed “chip”. Integrated circuits are available in several package styles. The two most common are the Dual In-line Package (DIP) and Surface Mount Technology (SMT) packages. Inside the integrated circuit package is a small silicon chip measuring less than 1/4-inch square. This chip contains the transistors and connecting circuits required for implementing the logic components.

The number of gates fabricated on the integrated circuit classifies integrated circuits. The classification is approximated by:

SSI (Small Scale Integrated) circuit:	1 to 10 gates
MSI (Medium Scale Integrated) circuit:	10 to 100 gates
LSI (Large Scale Integrated) circuit:	100 to 100,000 gates
VLSI (Very Large Scale Integrated) circuit:	> 100,000 gates

3.1. Dual In-Line Packages

A common integrated circuit package is the Dual In-line Package or DIP. Figure 3.1 illustrates mechanical views of a dual in-line package for a 14 pin DIP. DIPs are commonly available in 14, 16, 20, 22, 24, 28, 40, 64, and 68 pin arrangements with the pins always positioned in two parallel rows as shown in Figure 3.1. Package materials are usually plastic or ceramic, with the pins made of gold or tin plated metal. Electrical contact between the pins and the silicon chip is usually made using gold filament wires which are ultrasonically welded to conductive pads on the silicon chip. Proper orientation of the IC is determined by using either the notch shown in or by locating a small depression on the top of the IC, which specifies pin 1.

One very common SSI circuit family is the 7400 Series integrated circuits, which are produced by a variety of semiconductor manufacturers. Functional views of several of these integrated circuits are illustrated in Figure 3.2. These ICs contain the basic logic gates described in Section 2.1. Specific pins on the integrated circuit package are connected internally to the inputs and output of a gate, which has been fabricated on the chip. Power and ground are symbolized by V_{cc} and GND. Power usually comes from a 5 Volt source for most digital circuits. The following paragraphs describe four integrated circuit classifications and examples for each.

Small Scale Integrated Circuits (SSI) usually contain several gates inside one integrated circuit package. When describing the number of functional units in the integrated circuit, one usually uses the prefixes dual for two, triple for three, quad for four, and hex for six units in the package. For example the integrated circuits of Figure 3.2 would be described as follows:

7400 = Quad 2-Input NAND Gates,	7404 = Hex Inverters,
7411 = Triple 3-Input AND Gates,	7421 = Dual 4- Input AND Gates

To construct a digital circuit, integrated circuits are usually mounted on a printed circuit board. The ICs are interconnected using conductive wire-like circuit paths, which are etched on the circuit board when it is manufactured. The pins of the ICs and other electrical components are soldered to the circuit board to ensure good electrical contact and mechanical bonding.

Figure 3.1 14-Pin Dual Inline Package

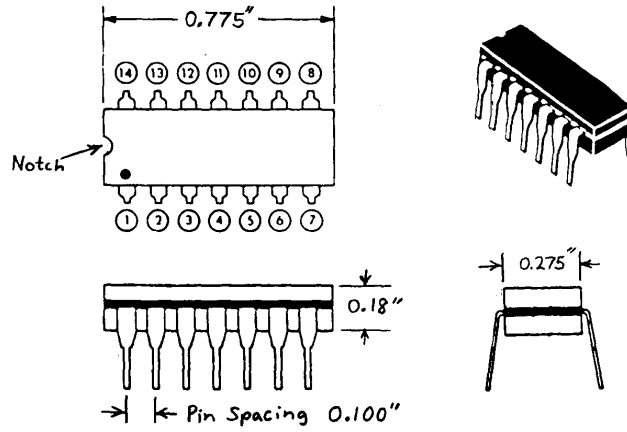
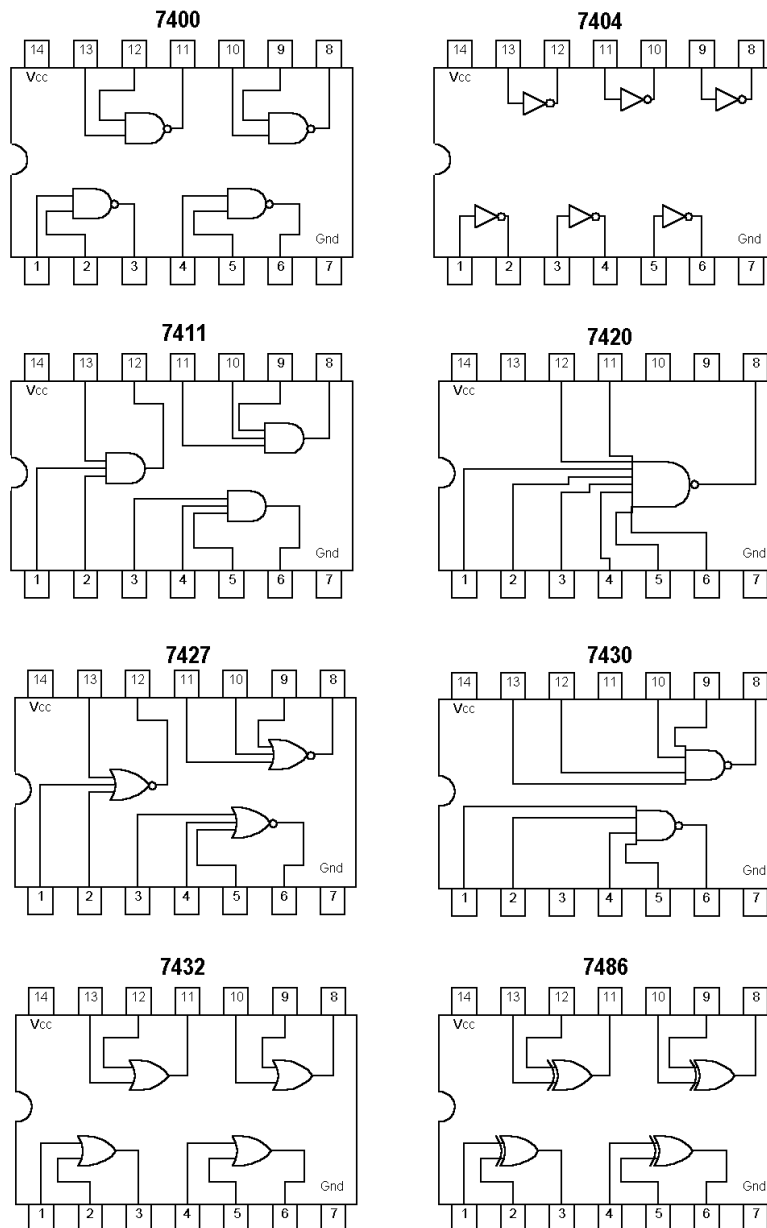


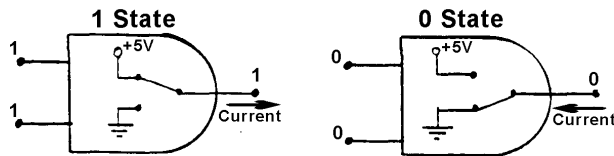
Figure 3.2 Functional Views of Several 7400 Series Integrated Circuits



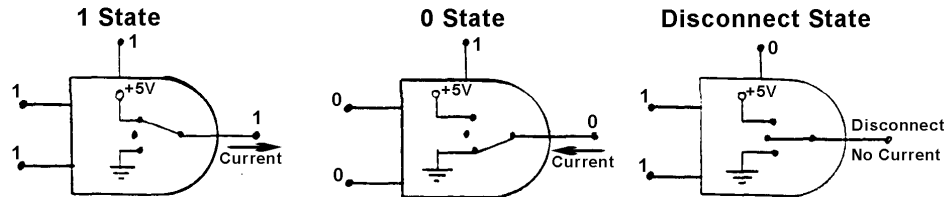
buses. When the output states of two connected totem-pole outputs are different, the low state output may damage the high state output. This problem is remedied by using either tri-state output gates.

Figure 3.4 Switch Analogies for Output Devices

a) Standard Gate Output



b) Tri-state Gate Output



3.2.1. Tri-state Gates

The tri-state gate has transistor switch outputs, which can be modeled as a 3-position switch as depicted in Figure 3.4b. Each position of the switch represents a different state (either 0, 1, or Disconnect). Notice that an additional input is provided on the top of the gate, which is called the control input. The control input is used to selectively disconnect or drive the output. The truth table for a tri-state NAND gate is shown in Figure 3.5a. When the Control input is 1, the gate functions normally as a 2-input NAND gate. However, when the Control input is 0, the gate output is effectively disconnected. This disconnect state is often referred to as the high-impedance state (High-Z State).

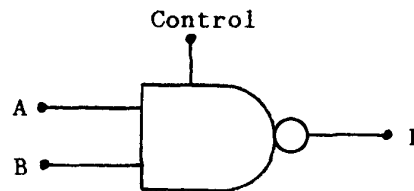
Figure 3.5b illustrates a tri-state buffer with a negative logic control input. The tri-state buffer is symbolized as a triangle pointing to the output (similar to the NOT gate without the inversion circle). As described by the truth table, the output of the tri-state gate is disconnected when the control input is in the 1 state. When the control input is in the 0 state, the output state is the same value as the input A.

When a bus is constructed using tri-state gates, only one tri-state gate can be active at a time. Inactive outputs must be disconnected from the bus by forcing their control inputs to the inactive state.

Figure 3.5 Tri-state Gate Truth Table and Symbolic Representation

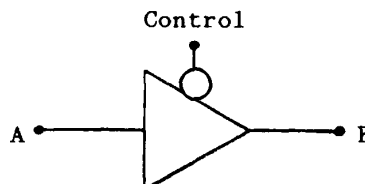
(a)

Control	A	B	F
0	0	0	Disconnect
0	0	1	Disconnect
0	1	0	Disconnect
0	1	1	Disconnect
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



(b)

Control	A	F
0	0	0
0	1	1
1	0	Disconnect
1	1	Disconnect

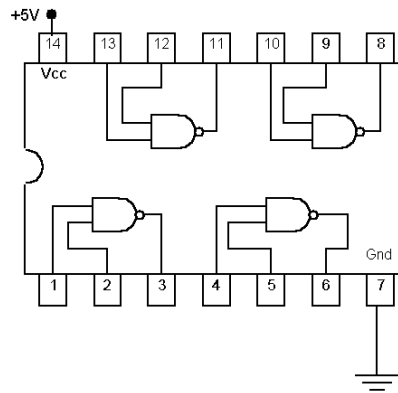
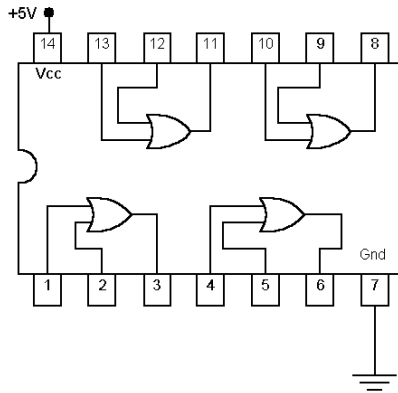


Problem Set

1. Using the integrated circuits shown, draw lines between the pins of the DIPs to construct a digital circuit to implement the Boolean expression

$X = A + \overline{B}C + \overline{D}$ for Output X based on inputs A, B, C, and D. The power and ground connections are drawn.

$X = A + \overline{B}C + \overline{D}$ for Output X based on inputs A, B, C, and D. The power and ground connections are drawn.



2. Using the integrated circuits shown in Figure 3.2, draw circuits for the following Boolean expressions.

- $A + BC$
- $A(B + C)D$
- $A\overline{B}C + B\overline{C} + A$
- $(A \oplus B) \oplus C$

3. Draw the logic gate symbol and truth table for the following tri-state gates.

- Tri-state Inverter
- Tri-state 2-Input OR Gate
- Tri-state 3-Input NAND Gate